Behavioural synthesis of asynchronous controllers: a case study with a self-timed communication channel

Alex Yakovlev, Frank Burns, Alex Bystrov, Albert Koelmans, Delong Shang
University of Newcastle upon Tyne

Rene Krenz
Royal Institute of Technology, Stockholm

ACiD–WG Workshop, München, Jan. 2002

Outline

- Motivation
- Design flow
- Two-level behavioural synthesis
- Direct translation from LPNs and STGs
- Communication channel case study
  - Specification, verification, controller synthesis, optimisation and performance
- Conclusion
Motivation

• Complex asynchronous controllers still cannot be designed fully automatically.
• Existing logic synthesis tools (cf. Petrify and Minimalist) can only cope with small-scale low level designs (state-space explosion, limited optimisation heuristics).
• Logic synthesis produces circuits whose structure does not correspond to their behaviour structure (bad for analysis and testing).
• Syntax-direct translation techniques may be a way forward but applied at what level?

Motivation

• Applying directly at front-end (cf. Tangram) guarantees design productivity but may produce slow circuits (control flow is driven by program syntax, not by natural operation sequencing).
• Ideally, front-end (HDLs) needs efficient simulation support and flexible and rigorous interface with behavioural back-end (labelled Petri nets, STGs) used for synthesis.
• The back-end must support compositionality and hierarchy (of HDLs) but offer sequencing paradigms (causality and concurrency) for high performance.
• Optimisations can be applied to back-end models.
• Direct translation of LPNs and STGs helps structural transparency between specification and implementation.
Motivation

• Implications to new research targets on:
  – Translation between HDLs and LPNs, FSMs, STGs, particularly formal underpinning of semantic links between front-end and back-end formats
  – New composition and decomposition techniques (incl. various forms of refinement and transformations) applied to LPNs/STGs/FSMs
  – New circuit mapping and optimisation techniques for different types of models (under various delay-dependence or relative time assumptions and different signalling schemes)
  – Combination of direct mapping with logic synthesis (eg. circuits with predictable latency)
HDL syntax directed mapping

```
do
if (X=A) then
  par
  OP1;
  OP2;
  rap
else
  seq
  OP3;
  OP4;
qes
if
od
```

Control flow is transferred between HDL syntax constructs rather than between operations

---

Two-level behavioural synthesis

```
do
if (X=A) then
  par
  OP1;
  OP2;
  rap
else
  seq
  OP3;
  OP4;
qes
if
od
```

High level control: Labelled Petri net (LPN)
Two-level behavioural synthesis

Low level control: Signal Transition Graphs (STG)

Data path 1

OP1r → OP1a
OP3r → OP3a
OP4r → OP4a
req1 → ack1

Data path 2

OP2r → OP2a
ack2 → req2

High-level control logic directly mapped from LPN

Basic David cell (DC)
Direct mapping of LPNs and STGs

Communication channel example

- A duplex delay-insensitive channel for low power and pin-efficiency proposed by Steve Furber (AIN'T'2002)
- Relatively simple data path (with handshake access via push and pull protocols)
- Sophisticated control (involves arbitration, choice and concurrency)
- Natural two-level control decomposition
- Requires low-latency (existing STG and BM solutions produce too heavy logic)
Channel Structure

N-of-M codes: dual-rail, 3-of-6, 2-of-7

Key Protocol Symbols (e.g. in dual rail):
- Start (01), Ack (10), Slave-Ack (11), Data (01 or 10)

Protocol Specification

The protocol can be defined on an imaginary Protocol Automaton receiving symbols from both sides (it will hide all activity internal to Master and Slave)
Protocol Specification

Controller Overview

Sender

Sending--Interface

Main Controller

High Level control

Data path and low level control

RxData RxLast RxCont R RxStart RxAck

RxData TxLast TxCont TxReq TxStart TxAck

RxData RxLast RxCont R RxStart RxAck

RxData

Main Controller

RxData

RxLast

RxCont

R

RxStart

RxAck

push

push

push

pull

pull

outgoing data channels

incoming data channels
Low-level logic

LPN model for high level control (master)

Calls to local arbiters
pulls
pushes
dummies inserted for direct DC mapping
Slave-Ack pull
Three-way pushes
Three-way pulls
High level control (master) mapped directly from LPN

Towards synthesis for higher performance

Is the dummy in the right place?
It is on the cycle of (output) push and (input) pull:
pull->dummy->push->pull-dummy->push->

...
Towards synthesis for higher performance

<table>
<thead>
<tr>
<th>Sender</th>
<th>Sending-Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>RxData</td>
<td>RxLast RxCont RxReq RxStart RxAck</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>RxData</td>
<td>RxLast RxCont RxReq RxStart RxAck</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver</td>
<td>Receiving-Interface</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TxData</td>
<td>TxLast TxCont TxReq TxStart TxAck</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TxData</td>
<td>TxLast TxCont TxReq TxStart TxAck</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Main Controller

Critical path

Synthesis rule:
Don’t insert dummies on critical paths

Synthesis for lower I/O latency
LPN level

<table>
<thead>
<tr>
<th>High-level control</th>
</tr>
</thead>
<tbody>
<tr>
<td>pull</td>
</tr>
<tr>
<td>push</td>
</tr>
<tr>
<td>internal actions</td>
</tr>
<tr>
<td>pull</td>
</tr>
</tbody>
</table>

... pull logic pull logic pull logic...

Environment (channel)

... input output input...
Channel Cycle Time

<table>
<thead>
<tr>
<th>Controller Implementation</th>
<th>Simplex mode</th>
<th>Duplex mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapping from LPN</td>
<td>7.6 ns</td>
<td>8.3 ns</td>
</tr>
<tr>
<td>Logic synthesis from STG</td>
<td>12.7 ns</td>
<td>16.5 ns</td>
</tr>
</tbody>
</table>

- These results were obtained for 0.6 micro CMOS
- Further improvement can be achieved by more use of low latency techniques (at the gate level) and introducing aggressive relative timing, in David cells and low level logic

Conclusion

- Hierarchical (eg. Protocol) controller synthesis can go via back-end LPN/STG models
- Direct mapping from LPNs/STGs yields fast circuits that are easy to analyse and test
- Translation from PNs to David cell netlists implemented in tool pn2dc
- Translation from FSM VHDL specs to LPNs and STGs implemented in tools fsm2lpn and fsm2stg
- Further work needed on:
  - Formal link between HDLs and PNs (semantics and equivalence), leading to better synthesis of PNs from HDLs
  - Optimisation techniques at LPN/STG and circuit levels
- See our papers in Async’02 and 11th UK Async Forum
Design flow

- What is now being developed at Newcastle?
  - translation from ‘subset VHDL’ (and other languages) to LPNs and STGs
  - direct synthesis from LPNs and STGs
  - combined direct and logic (Petrify) synthesis
  - optimisation at LPN/STG level (eg. for low latency)

Direct mapping of LPNs and STGs to David Cell netlist

Operation can be interpreted as access to datapath (LPN) or as switching a binary (input or output) signal (STG)
Protocol Refined (for Dual Rail encoding)

Protocol Verification

Properties to be verified:
absence of deadlock and delay-insensitivity (w.r.t. delays in the channel wires)
Protocol Verification

Petri net model of the protocol for verification

Fragment of the master subnet for verification

These places must be 1-safe to have freedom from communication interference (delay-insensitivity)

Protocol Verification

The Petri net unfolding prefix was constructed by tool PUNT and checked:
There are no deadlocks
The net is 1-safe w.r.t. channel places (which proves delay-insensitivity)